Learning to Rank Research Articles

A case study of collaborative filtering and learning to rank in ScienceDirect

Daniel Kershaw, Benjamin Pettit, Maya Hristakeva, and Kris Jack

Elsevier Ltd

Abstract. Online academic repositories help millions of researchers discover relevant articles, a domain in which there are many potential signals of relevance, including text, citation links, and how recently an article was published. In this paper we present a case study of prodictionizing learning to rank for large scale recommendation, which utilises these diverse feature sets to increase user engagement. We first introduce item-to-item collaborative filtering (CF), then how these recommendations are rescored with a LtR model. We then describe offline and online evaluation, which are essential for productionizing any recommender. The online results show that learning to rank significantly increased user engagement with the recommender. Finally we show through post-hoc analysis that the original CF solution tended to promote older articles with lower traffic. However, by learning from subjective user interactions with the recommender system, our relevance model reversed those trends.

1 Introduction

The rate of scientific discovery is ever increasing, with new methods, theory and practice being published each day. This growth poses a challenge for researchers, who need to stay up to date. Online academic catalogues, such as *ScienceDirect*,[1] give users access to large amounts of peer reviewed scientific publications. However, the experience of using such catalogues can be characterised as a combination of *information overload* [4] and *information shortage* [12]. First, when encountering a large catalogue of information there is no simple way for the user to read, comprehend and critique all the documents. Additionally, when browsing they may not be discovering content that they would deem relevant.

The specific use case we aim to address is to help users of *ScienceDirect* find additional relevant articles that go beyond explicitly encoded relationships such as authorship or publication venue. In contrast to the personalised research article recommendations in platforms such as Mendeley [13] and CiteULike [19], we want an approach that works in the absence of profile data or reading history.

[1] `https://www.sciencedirect.com`

In this paper, we describe how we built an initial system using item-based collaborative filtering (IBCF). Once this system is in production, we collected data on which recommendations users preferred, and then trained a Learning to Rank (LtR) model to re-rank the recommendations using a range of article features and similarity metrics. We focus on the evaluation methodology and how the recommendations surfaced by LtR differed from the pure collaborative filtering system. These give insight into what the LtR system achieves that could not be done with collaborative filtering (CF) alone.

Recommender Systems (RS) have become key tools in a researcher's content discovery toolkit, as they not only allow them to more efficiently navigate large and ever-growing catalogues, but also discover content that they would not have seen otherwise. Previous work has resulted in a distinction between methods used for personalised and non-personalised approaches. This can be seen in the use of implicit feedback and CF for personalised recommendations [13, 19], whereas content-based methods are used predominately for non-personalised recommendations [20, 21, 16]. In combining CF and LtR we overcome some of the limitations of CF by reducing the dependence on current navigation patterns, allowing us to take into account the content of the article [7] and how recently it was published. Unlike purely content-based recommendations, this system adapts to how the articles are being used and which recommendations users engage with.

Through this research, we ultimately found that to train and evaluate a better model of article relevance, it is necessary to go beyond trying to predict what users will browse next. While the implicit feedback from article browsing is valuable for CF, it is limited by the very same information shortage problem that the recommender system attempts to mitigate. In addition to developing the model, we present some post-hoc analysis that sheds light on some of the biases in the CF results that are reversed by applying LtR. Our contributions can be summarised by the following points:

1. **Offline evaluation should be matched to the online challenge**: By comparing two offline evaluation scenarios with an online experiment, we show that higher accuracy at predicting browsing behaviour does not correspond to having the highest engagement from users.
2. **The winning ranking model depends on text, usage, article age, and the citation network**: By training a number of LtR models we show that relevance of a document to a user is a combination of textual similarity, recency, popularity, usage similarity, and proximity in the citation network. Traditional bibliometric features have limited impact.
3. **The ranking model increases diversity**: On average, the ranking model increases the number of distinct journals in each list of related items.
4. **The ranking model promotes recently published items that have more traffic in the past year:** Although other collaborative filtering systems have reported a popularity bias [2], in this case collaborative filtering has a bias towards unpopular items, which the LtR system reverses.

2 Related Work

A wide variety of techniques have been used to generate recommendations for academic publications. [21] and [20] focused on using hierarchical clustering of citation networks to make recommendations that distinguish between core papers in a discipline and important papers in sub-fields. While, [10] and [18] used author-defined keywords and tags to identify similar documents. Not only explicit article metadata are used to generate recommendations: Mendeley Suggest [13][2] used user-based collaborative filtering (UBCF) to generate recommendations based on a user's library of documents, which showed improvements over content and citation based methods. Another example comes from Wang *et al.* [19], who used implict feedback from CiteULike[3] in conjunction with Latent Dirichlet allocation (LDA) applied to article text. For a comprehensive survey of research article recommender systems, we refer the readers to [3].

LtR has traditionally been used in information retrieval (IR) systems such as search engines [14]. For example, WalMart demonstrated that LtR could be used to improve the rankings of Grocery search results [15]. They used features mined from images of the products, and experimented with optimising the models for different actions such as clicking on the item or buying the item.

To aid research into LtR, a number of frameworks have been developed which allow for the test and training of models. Microsoft released LEROT [1], an "online learning to rank framework". Additionally, RankLib [8] is a `JAVA` framework that contains a number of standard LtR models, which will be discussed next. These frameworks have allowed research to be reproducible and applied easily across a number of domains. These frameworks implement a number of different algorithms between point-wise, pair-wise and list-wise approaches.

3 Data sets

The recommendations generation has two distinct phases. First we generate the recommendation candidates using CF and then re-rank them with a LtR model to get the recommendations list. We use different data sets within these two distinct phases.

3.1 Article browsing logs

The main data set for CF contains implicit feedback from users as they browse *ScienceDirect*. This usage data set is in the format of `<sessionID, articleID, accessTime>`. We apply IBCF on this data set to find candidate related articles based on co-usage patterns. High traffic users are removed, as these can represent public access machines in institutions. Additionally, we remove traffic that was elicited by the recommender system, in order to remove a positive feedback loop.

[2] `https://www.mendeley.com/suggest`
[3] `https://www.citeulike.com`

3.2 Recommender logs

LtR requires labelled training data that represents user preferences in relation to the recommendation list. We computed relevance labels by aggregating clicks and impressions from the live CF recommender on *ScienceDirect*. When users visit a *ScienceDirect* article page, a set of related-article recommendations are displayed (i.e. impressions) and they may click on one or more of the recommendations to view or download the article. We ignored impression data for page loads where none of the recommendations were clicked, or where all of the recommendations were clicked. For each query article, we aggregated the recommended articles across all user sessions. For simplicity we treat relevance as a binary label: 1 if the recommended article was clicked at least once, otherwise 0.

3.3 Article metadata

Each article has a large amount of data and meta-data associated with it. This includes titles, authors, abstract, references, and various metrics on article, journal impact and citation information. These additional data sets are used to generate features for the LtR models.

4 Recommendation Method

4.1 Collaborative Filtering

At the core of IBCF is the method proposed by [17], which finds similar items (documents) based on the similarity between their usage vectors. This means that for a given document (d) the function $sim(d, D)$ returns a set of M similar documents based on their co-usage patters. Within this nearest-neighbour method, we use cosine similarity to identify documents that have been browsed in the same sessions. Here, S_d is the set of sessions in which document d was browsed.

$$cosine(d, d') = \frac{|S_d \cap S_{d'}|}{\sqrt{|S_d| \times |S_{d'}|}} \tag{1}$$

Using cosine similarity to score neighbours ignores the statistical confidence in the correlation between usage patters. For example, many documents were not viewed in very many sessions, so a similar document d' may have only one or two sessions in common with the focal document d, but nonetheless rank highly in terms of $cosine(d, d')$ because $|S_{d'}|$ is also small. Therefore, we scale cosine similarity with a significance weighting computed from the number of articles in common:

$$score(d, d') = \min(1, \frac{|S_d \cap S_{d'}|}{q}) \times cosine(d, d') \tag{2}$$

If the documents have fewer than q sessions in common, then its contribution to the CF score is scaled down. This means that preference is given to recommendations that are generated from high co-occurrence neighbours, who are more likely related to the focal document d. An alternative would be to discard pairs with low co-occurrence, but to keep catalogue coverage high we chose to keep them and reduce their scores instead.

4.2 Learning to Rank

Once the set of recommendations (R_d) have been generated for each document ($d \in D$) it is re-scored using a pre-trained LtR model. The premise of LtR is to rank items higher which users are more likely to engage with through observing their past action. Training takes the form of n labelled query documents, $q_i (i = 1, ..., n)$ and their associated recommended documents with feature vectors with relevance judgements $\mathbf{x}^i = \{x_j^{(i)}\}_{j=1}^{m^{(i)}}$, where n is the number of recommendations in the query.

For this work we focus on methods which have been implemented in the LtR package RankLib [8]. This is a `JAVA` application which can apply popular LtR methods to an SVMRank formatted file. The algorithms we compared included RankNet, LambdaRank, MART, and LambdaMART. These LtR models represent both pair-wise and list-wise objectives.

RankNet [5] is a pair-wise neural network algorithm. The objective function is cross entropy, which aims to minimise the number of inversions in the ranking. However, this pair-wise objective does not optimise for the whole list, unlike list-wise methods such as LambdaRank and LambdaMART. LambdaRank [6] built on RankNet by modifying the cost function to use gradients, λ, which also take into account the change in a list-wise IR metric such as NDCG. LambdaMART [6] combines LambdaRank and multiple additive regression trees (MART), using the cost function from LambdaRank rather than from MART, thus optimizing for the whole list. Out of the available models, LambdaMART is generally considered state-of-the-art, and has performed well in competitions [6].

Feature Extraction The features for the LtR models are taken from the query document and the recommended document as well as from the interaction between the two. These features can be grouped into seven categories.

CF score: A measure of co-usage of the query document and the candidate recommendation (Equation 2).

Popularity: Popularity (number of views) of a document potentially indicates its quality or its future engagement.

Citations: Two documents (the query document and the recommendation) share references then this could indicate a quality recommendation, and likewise if two articles are both cited by the same article. To quantify this, we compute two measures, the first being the Jaccard index between the citation neighbourhoods,

$$cite\_sim(C_d, C_{d'}) = \frac{|C_d \cap C_{d'}|}{|C_d \cup C_{d'}|} \quad (3)$$

where the neighbourhood C_d is the set of articles that either cite document d or are cited by document d, plus document d itself. The second measure is the total number of citations a document has received.

Journal Metrics: Impact factors are potential predictors of the quality of the research, although a weakness is the huge variation in article impact within a journal. We added to the feature set several impact metrics of the journal where the recommended article was published .

Temporal: We represent age as the number of years since the cover date.

Topics: All articles published are tagged with a scientific taxonomy, indicating which topics and subjects they cover. We calculate binary similarity between the sets of topics associated with each document .

Text: We included the similarity of the recommended document text to the query document text, where text is represented as an n-gram tf-idf vector of a document's title and abstract.

Training We used query-recommendation pairs with relevance labels inferred from recommender logs, as described in Section 3.2. We held out 20% of the query articles from the training data as a validation set, and trained on the remaining 80%. The validation set was used for hyper-parameter tuning and feature selection.

For hyper-parameter tuning, LambdaMART and MART require choosing the learning rate, the number of trees, maximum leaves per tree, and minimum training examples per leaf. With LambdaMART, we used 20 leaves per tree, with at least 200 examples per leaf and a learning rate of 0.1. The number of trees (~ 250) was determined by early stopping, again based on the validation set. The final model used was LambdaMART, optimising for `NDCG@3`.

We pruned features through backwards elimination: removing the least important feature (highest `NDCG@3` when removed), and then repeating the process as long as `NDCG@3` increased on the validation set. In Section 6.3 we compare the importance of the different types of features.

4.3 Dithering

The candidate recommendations are ranked using the LtR model or, in the initial version of the system, by their CF scores. Before selecting the top portion of the list to display to the user, we apply *dithering* [9], so that a larger proportion of the list is explored. Dithering is the process of adding Gaussian noise to the items' ranks, thus slightly shuffling the list; $new\_score = \log(rank) + N(0, \log \varepsilon)$, where $\varepsilon = \frac{\Delta rank}{rank}$ and typically $\varepsilon \in [1.5, 3]$. Over time, items at lower ranks will eventually be shown to users, allowing us to collect feedback on their quality as recommendations.

Table 1. Comparison of LtR models using click prediction task on both the time-split test set and validation set.

Ranker	NDCG@3 Validation	NDCG@3 Test
Random	0.753	0.768
CF score	0.802	0.787
LambdaMART	**0.814**	**0.807**
MART	0.813	0.804
RankBoost	0.805	0.797
AdaRank	0.802	0.787
Random forests	0.808	0.798

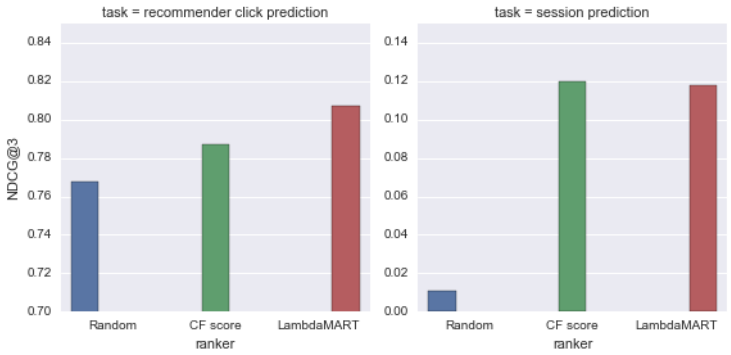


Fig. 1. LambdaMART performance on two offline evaluation tasks, compared to ranking by CF scores and a random baseline.

This is important because the LtR model is trained on impressions and clicks of recommendations (Section 3.2). Dithering makes the training data for LtR less constrained by the previous iteration of the recommender system.

5 Evaluation Method

We used two different tasks to evaluate LtR offline. First, we test it on recommendation click prediction, the same task that the model was trained on. Then we test whether the model transfers to a session prediction task.

In the *recommendation click prediction* task, each test case consists of a query article and its recommended articles that were displayed to users. Some of the recommendations were clicked (labelled 1) and some were not clicked (labelled 0). The recommender is evaluated on its ability to rank the clicked items higher than the non-clicked, using NDCG@k. This is the same ranking task that the LtR model was trained on, but the test data came from a time interval after the training and validation data. Its limitation is that it only evaluates performance

on the recommendations that were displayed by the incumbent recommender system, so it cannot judge articles that a new ranker will introduce into the top k.

In the *session prediction* task, each test case consists of a sequence of articles that were browsed in the same user session, excluding browsed articles elicited by the recommender system. The top k recommendations for the first item are compared to what the user actually browsed next. The recommender is evaluated on its ability to rank the browsed items highly (`NDCG@`k). We have found this evaluation procedure useful for tuning CF candidate selection, where it correctly predicted which CF variant had higher click through rate (CTR).

For both types of evaluation, we split the data set on a time boundary (τ), train the model on data generated before τ, and then test the model to see if it predicts the actions after τ. This means before τ we use the clicks in sessions (browsing between articles) to train the CF model, and then clicks on the already deployed CF recommender is used to train the LtR model. We use actions after τ as the ground truth, where sessions browsed are used for sessions prediction and click on the deployed recommender for click predictions. Although they were collected in the same time window, the test sets from the two evaluation tasks use mutually exclusive subsets of the logs, e.g. the first only uses recommender clicks, whereas the second excludes recommender clicks.

We calculated a random baseline for each evaluation task. For session prediction, the baseline is a random permutation of the candidate recommendation list for each query article. For recommendation click prediction, the baseline is a random permutation of the displayed recommendations for each query article.

6 Results

6.1 Offline evaluation

LambdaMART performed best in terms of `NDCG@3` on a time-split test set of recommender clicks and impressions (Table 1).

Having chosen a modelling approach for LtR, we tested whether LambdaMART's performance gains transferred to the session-prediction task. As shown in Figure 1, the ranking using LambdaMART was an improvement over CF score on the click prediction task, but not on the session prediction task. The LambdaMART and CF score rankings outperformed the random baseline in both tasks (Figure 1). Nonetheless, we decided to proceed with online evaluation, because the click prediction task is based on user behaviour in the recommendation context, and we therefore thought it would be a better proxy for online performance. We discuss the discrepancy between the two offline evaluation tasks in more detail in Section 7.

6.2 Online evaluation

Our best candidate algorithms from offline evaluation are compared against the current production model via A/B testing. Prior to the work on LtR, we had

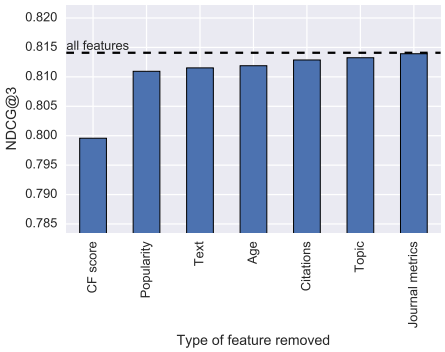


Fig. 2. Impact of removing each category of feature from LambdaMART model

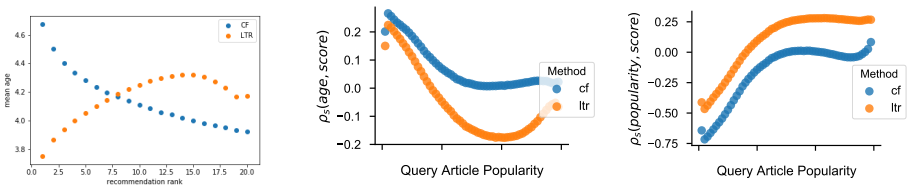
A/B tested a number of iterative improvements on the CF stage, for example by varying the duration of log data ingested to generate the recommendations. A/B tests performed included increasing the time period of input data for CF candidate selection which resulted in a CTR change of +9.6%, and Re-rank using LambdaMART mode which increase CTR by +9.1%.

For LtR, we carried out an online A/B experiment comparing LtR (using the LambdaMART model in Figure 1) to our best CF algorithm. Each user is randomly allocated to one of two treatments, either receiving recommendations from the incumbent CF system or from the LtR variant. The LtR variant resulted in a statistically significant 9.1% increase in CTR. This result agreed with the offline performance of LtR on the click prediction task, but it disagreed with their performance on the session prediction task (Figure 1).

6.3 Feature importance

Figure 2 shows the effect of removing each category of features, as introduced in Section 3. Using backwards elimination (Section 4.2), we obtained a slight increase in the validation metric (`NDCG@3`). Using fewer features also reduced training time and simplified the feature extraction pipeline. The journal metrics were all removed, and we retained one or two features from each of the other categories. The evaluation steps described above used this tuned LambdaMART model with a reduced feature set.

As one can see in Figure 2, CF score was the most important feature, in that removing it resulted in the greatest drop in `NDCG@3` on the held-out validation set. The feature importance for our ranking model should not be interpreted as what makes an article "relevant" in general, because the training data were collected through a specific user interface that displayed some article metadata and not others, and furthermore the order of the articles was determined by CF score. Instead, the impact of features within the model guided us as to which could be safely discarded (e.g. journal impact metrics).



(a) Mean age of items at a given rank.

(b) Correlation between age and rank

(c) Correlation between popularity and rank

Fig. 3. Post-hoc analysis results

6.4 Post-hoc analysis

Item Age One of the main motivations behind research article discovery tools is to help people stay up to date with recent developments in their fields. Therefore we examined the mean age of recommended documents in each rank position, where age is defined as the current year minus the publication year of the recommend article. As shown in Figure 3a, recommendations ranked by CF scores are biased against newly published articles, with higher-ranking articles tending to be older. When ranked by the LtR model, there is a bias towards younger articles instead.

The bias towards older articles, when ranked by CF, is especially pronounced where the query article is relatively *cold*, in other words it had less usage data available for collaborative filtering (Figure 3b). Figure 3b bins the query items based on their popularity within the CF input data, then for each query item it computes the Spearman's rank correlation (ρ_s) between the recommendation age and rank (ordered either by LtR or CF score), and finally takes the mean of ρ_s within each bin. A positive value of ρ_s indicates that older recommended articles tend to be ranked higher.

However, for recommendations ranked by LtR this effect of bias towards older articles has been reduced (lower values of ρ_s in every bin), resulting in warm articles having recommendations that favour newer publications.

Popularity In addition to favouring older articles, our CF method also tends to give higher scores to less popular articles (Figure 3b), where popularity is measured as the number of unique users viewing the item in the 12 months of input data for CF (this is assessed using the same method as in Section 6.4). This is the opposite pattern to the popularity bias often found in UBCF [2]. Like the age bias, it is most pronounced for colder source articles in the bottom two quartiles of popularity. LtR reverses this trend and instead biases the recommendations towards more popular items. Although this is not necessarily a desirable outcome for all recommender systems, it is not so surprising in this case, where popularity metrics were available as a feature for the relevance model. Despite the bias towards popular items, LtR caused only a 2.1% reduc-

Table 2. Journal diversity

	% of Query Articles	
Distinct Journals@3	CF score	LambdaMART
1	13.78%	12.27%
2	29.65%	32.16%
3	56.57%	55.57%

tion in `item coverage`@3, with over 90% of recommendable items appearing in the top three recommendations for at least one query article.

Diversity One of the challenges of RS is to highlight content from across a catalogue. Diversity in recommendations has been shown to increase users' interactions and perception of fairness [2]. However one of the issues with our system is that traffic can be concentrated within the same journal, which leads to some recommendation lists being all from the same journal. Thus we aim to understand how LtR and CF affect diversity within the set of recommendations.

To quantify diversity within our recommendations we look at the number of recommendations that come from different journals. With journals identified as distinct `ISSN` numbers.

As one can see in Table 2, LtR caused a slight increase diversity, in terms of the number of different journals in the top three recommendations for each query article. To further confirm the significance of change in diversity we perform a chi-square test (χ^2) of the contingency table showed a significant dependency between the ranker and number of distinct journals ($p < 10^{-16}$).

7 Discussion

In this paper we set out to show how LtR is applied to `item-to-item` academic paper recommendations. In doing so we were able to significantly improve user engagement with the RS, using a model trained on user interaction with the recommender system. The model learned how to combine many indicators of article relevance, including co-usage, popularity, age, text, topic tags, and shared citations. As with previous studies, we found that gradient boosted decision trees (MART, LambdaMART) performed well for this task [6], and that list-wise optimisation of the LambdaMART model was an additional advantage.

However, when it came to predicting what users would browse next, in the absence of recommendations, the model did not improve on IBCF. This is surprising, because we had previously found session prediction to be a good indicator of online performance, when tuning the collaborative filtering system. Recommendation is often framed as a problem of session prediction, for example in session-based recommendations with recurrent neural networks [11], but we found it had shortcomings.

Designing offline evaluation raises the question of what the recommender system is designed to do. The session-prediction task assumes the goal of recommendation is to predict human browsing behaviour. Although it was a good proxy for online performance of the candidate selection step, the session-prediction task did not agree with online results when it came to choosing a ranking method.

One explanation for why CF outperformed the LtR model on the session prediction task is that users' browsing behaviour outside of the recommender system is at odds with what they actually find useful as recommendations. Users might struggle to find the most relevant articles, which is the very problem of information shortage that motivated us to create the recommender. For example, a recently published article may be less well known within its field, and yet when it is presented in a recommendations list it attracts more attention than older articles.

The post-hoc analysis demonstrated some trends in the types of items recommended by IBCF, which LtR counteracted. Cold start and data sparsity are well-known challenges in collaborative filtering. Even among *warm* items with usage data, those with fewer users are expected to have less accurate recommendations. We found that the less popular the query article, the more the CF system tended to promote old or unpopular articles as recommendations. Based on which articles users clicked, the LtR system learned to promote articles that were published more recently or had more activity in the past year.

Based on these results, we should be able to improve the model by including as a feature the popularity of the query article (not just the recommended article). That would allow the tree-ensemble model to learn different relevance functions depending on how warm the source article is, for example giving less importance to CF score for items with sparser usage data. In theory, the model could also counteract the unpopularity-bias for colder articles without adding a popularity bias to the warmer articles' recommendations.

8 Conclusion

In this work we have demonstrated the benefits of LtR for improving upon a production CF system, which is especially important in a domain such as research articles where there are vast catalogues of recommendable items, each of which has high quality structured metadata. We showed how changes to the production system were ultimately decided on the basis of online evaluation, but given the number of models and parameter choices, we also needed an offline evaluation that was a good proxy for online performance.

The winning LambdaMART model combines co-usage, semantic similarity, shared citations, popularity, and recency. The original IBCF solution tended to promote older articles with lower traffic, but by learning from subjective user interactions with the recommender system, our relevance model reversed those trends and therefore overcame some of the limitations of CF.

References

1. S. Whiteson M. de Rijke A. Schuth, K. Hofmann. Lerot: an online learning to rank framework. In *Living Labs for Information Retrieval Evaluation workshop at CIKM'13.*, 2013.
2. Himan Abdollahpouri, Robin Burke, and Bamshad Mobasher. Controlling popularity bias in learning-to-rank recommendation. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, RecSys '17, pages 42–46, New York, NY, USA, 2017. ACM.
3. Joeran Beel, Bela Gipp, Stefan Langer, and Corinna Breitinger. Research-paper recommender systems: a literature survey. *International Journal on Digital Libraries*, 17(4):305–338, Nov 2016.
4. Christian Borgs, Jennifer Chayes, Brian Karrer, Brendan Meeder, R. Ravi, Ray Reagans, and Amin Sayedi. Game-theoretic models of information overload in social networks. In Ravi Kumar and Dandapani Sivakumar, editors, *Algorithms and Models for the Web-Graph*, pages 146–161, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.
5. Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22Nd International Conference on Machine Learning*, ICML '05, pages 89–96, New York, NY, USA, 2005. ACM.
6. Christopher J. C. Burges. From ranknet to lambdarank to lambdamart: An overview. 2010.
7. Fidel Cacheda, Víctor Carneiro, Diego Fernández, and Vreixo Formoso. Comparison of collaborative filtering algorithms: Limitations of current techniques and proposals for scalable, high-performance recommender systems. *ACM Trans. Web*, 5(1):2:1–2:33, February 2011.
8. Van Dang. RankLib, 2011.
9. Ted Dunning, Ellen Friedman, and M D Ellen Friedman. *Practical Machine Learning: Innovations in Recommendation.* O'Reilly Media, Inc., August 2014.
10. Felice Ferrara, Nirmala Pudota, and Carlo Tasso. A keyphrase-based paper recommender system. In Maristella Agosti, Floriana Esposito, Carlo Meghini, and Nicola Orio, editors, *Digital Libraries and Archives*, pages 14–25, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
11. Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. *CoRR*, abs/1511.06939, 2015.
12. Liangjie Hong, Ron Bekkerman, Joseph Adler, and Brian D. Davison. Learning to rank social update streams. In *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '12, pages 651–660, New York, NY, USA, 2012. ACM.
13. Maya Hristakeva, Daniel Kershaw, Marco Rossetti, Petr Knoth, Benjamin Pettit, Saúl Vargas, and Kris Jack. Building recommender systems for scholarly information. In *Proceedings of the 1st Workshop on Scholarly Web Mining*, SWM '17, pages 25–32, New York, NY, USA, 2017. ACM.
14. Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 133–142, New York, NY, USA, 2002. ACM.
15. Shubhra Kanti Karmaker Santu, Parikshit Sondhi, and ChengXiang Zhai. On application of learning to rank for e-commerce search. In *Proceedings of the 40th*

International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '17, pages 475–484, New York, NY, USA, 2017. ACM.
16. Jimmy Lin and W. John Wilbur. PubMed related articles: a probabilistic topic-based model for content similarity. *BMC Bioinformatics*, 8(1):423, oct 2007.
17. Greg Linden, Brent Smith, and Jeremy York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, January 2003.
18. Cristiano Nascimento, Alberto H.F. Laender, Altigran S. da Silva, and Marcos André Gonçalves. A source independent framework for research paper recommendation. In *Proceedings of the 11th Annual International ACM/IEEE Joint Conference on Digital Libraries*, JCDL '11, pages 297–306, New York, NY, USA, 2011. ACM.
19. Chong Wang and David M. Blei. Collaborative topic modeling for recommending scientific articles. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, pages 448–456, New York, NY, USA, 2011. ACM.
20. Ian Wesley-Smith and Jevin D. West. Babel: A platform for facilitating research in scholarly article discovery. In *Proceedings of the 25th International Conference Companion on World Wide Web*, WWW '16 Companion, pages 389–394, Republic and Canton of Geneva, Switzerland, 2016. International World Wide Web Conferences Steering Committee.
21. J. D. West, I. Wesley-Smith, and C. T. Bergstrom. A recommendation system based on hierarchical clustering of an article-level citation network. *IEEE Transactions on Big Data*, 2(2):113–123, June 2016.